〈[파이썬 알고리즘 탐험] 제2장. 소수들의 숨겨진 패턴과 비밀 암호!〉

선생님: (여러 개의 숫자 카드를 보여주며) 얘들아, 안녕! 우리가 지난 시간에 만든 `is_prime()` 함수 기억나니? 그 함수는 어떤 숫자가 1과 자기 자신만을 약수로 가지는 '소수'인지 아닌지를 `True`나 `False`로 똑 부러지게 알려줬지!

코딩새싹 🌱: 네! 제곱근까지만 나눠보는 똑똑한 탐정이었어요!

선생님: 맞아! 그런데 만약 우리가 "어떤 숫자까지의 모든 소수를 찾아내고 싶다!"면 어떻게 해야 할까? 여기, 그런 시도를 한 함수가 있단다. 한번 같이 볼까?

〈1. 소수 모으기, 첫 번째 시도! (어딘가 이상한데…?)〉

```
def primes_version1(num): # num까지의 모든 소수를 모으고 싶어!
  result = []
  for i in range(2, num + 1): # 2부터 num까지의 숫자를 하나씩 보면서...
    # i가 소수인지 여기서 판별해야 하는데...
    for x in range(2, i): # 2부터 i-1까지의 숫자로 나눠보려고 시도!
      if i % x == 0: # 만약 i가 x로 나누어떨어지면, i는 소수가 아니지!
        pass         # '일단 통과!' ... 어?
    result.append(i) # 그리고 i를 결과 리스트에 추가한다...?
  return result

# print("primes_version1(10)의 결과:", primes_version1(10))
```

궁금이 💡: (코드를 유심히 보며) 선생님! 저 `pass`라는 건 '아무것도 안 하고 그냥 지나간다'는 뜻 아니에요? 만약 `i`가 4일 때, 안쪽 `for`문에서 `x`가 2가 되고, `4 % 2 == 0`이 되니까 `pass`를 하고… 그러고 나서 바깥쪽 `result.append(4)`가 실행되면… 어? 4는 소수가 아닌데 리스트에 들어가 버리는데요?

선생님: (빙긋 웃으며) 바로 그거야, 궁금아! 예리한 눈으로 문제점을 정확히 찾아냈구나! 이 `primes_version1` 함수는 안타깝게도 안쪽 `for`문에서 소수인지 아닌지를 판별하는 로직이 완성되지 않았어. `pass`는 아무것도 하지 않기 때문에, 바깥쪽 `result.append(i)`는 `i`가 소수이든 아니든 무조건 실행되어서 2부터 `num`까지 모든 숫자를 담게 된단다.

코딩새싹 🌱: 아하! 코드가 에러 없이 돌아간다고 해서 항상 우리가 원하는 대로 동작하는 건 아니군요!

선생님: 정말 중요한 깨달음이야! 그래서 우리는 지난 시간에 만든 똑똑한 탐정, `is_prime()` 함수를 활용해야 하는 거란다!

<2. 소수들 사이의 가장 큰 틈새를 찾아라!>

선생님: 자, 그럼 우리의 훌륭한 `is_prime()` 함수를 사용해서 2부터 1000까지의 소수들을 먼저 쫙~ 모아보자.

```
# (지난 시간에 만든 is_prime 함수가 있다고 가정!)
def is_prime(num):
  if num < 2: return False
  for i in range(2, int(num ** 0.5) + 1):
    if num % i == 0: return False
  return True

# 2부터 1000까지 소수만 모으기
primes_list = []
for i in range(2, 1001):
  if is_prime(i):
    primes_list.append(i)

print("--- 2부터 1000까지의 소수들 (일부) ---")
print(primes_list[:20]) # 너무 많으니까 앞 20개만 살짝!
print(f"(총 {len(primes_list)}개의 소수를 찾았어요!)")
```

선생님: 이제 이 소수들 목록(`primes_list`)에서, 연속된 두 소수 사이의 간격(차이)이 가장 큰 곳은 어디일까? 그 간격은 얼마일까?

코딩새싹 🌱: 음… 소수들을 하나씩 보면서 바로 옆에 있는 소수와의 차이를 계속 비교해보면 될 것 같아요!

선생님: 좋은 생각이야! 가장 큰 차이를 기록할 변수와, 그 위치를 기억할 변수가 필요하겠지?

```
delta = 0 # 지금까지 찾은 가장 큰 차이를 저장할 변수 (처음엔 0으로 시작)
a = 0     # 가장 큰 차이를 만든 두 소수 중 두 번째 소수의 "인덱스"를 저장할 변수

# 소수 리스트에서 두 번째 소수부터 시작해서 바로 앞의 소수와 비교해야 해!
```

```
# 그래서 range(1, len(primes_list)) 로 두 번째 소수부터 마지막 소수까지 반복!
for i in range(1, len(primes_list)):
  current_diff = primes_list[i] - primes_list[i-1] # 현재 소수와 바로 앞 소수의 차이
  if current_diff > delta: # 만약 이 차이가 지금까지 기록된 가장 큰 차이(delta)보다 크다면,
    delta = current_diff # 기록을 새롭게 갱신!
    a = i                # 그리고 그 위치(두 번째 소수의 인덱스)를 기억!

print(f"\n가장 큰 차이는 {delta} 입니다.")
print(f"그 구간은 {primes_list[a-1]} ~ {primes_list[a]} 입니다.")
```

궁금이 💡: primes_list[i-1] 이랑 primes_list[i] 로 바로 옆에 있는 두 소수를 비교하는 거군요! delta라는 변수에 계속해서 더 큰 차이값을 업데이트하니까 가장 큰 차이를 찾을 수 있네요!

〈3. 숫자를 소인수분해 하다! (숫자의 비밀 암호 풀기!)〉

선생님: 모든 합성수는 소수들의 곱으로 유일하게 나타낼 수 있다는 사실, 알고 있니? 예를 들어 12는 2 × 2 × 3 이지. 이렇게 어떤 숫자를 소수들의 곱으로만 나타내는 것을 **소인수분해**라고 해. 마치 숫자의 비밀 암호를 푸는 것과 같단다!

코딩새싹 🌱: 와! 그럼 어떤 숫자를 주면, 그 숫자를 이루는 소수들을 찾아주는 함수를 만들 수 있나요?

선생님: 물론! factorization(숫자) 함수를 만들어보자!

```
def factorization(n):
  result = [] # 소인수들을 담을 빈 리스트
  divisor = 2   # 나누는 수(소인수가 될 후보)는 가장 작은 소수인 2부터 시작!

  while divisor <= n: # 나누는 수가 원래 숫자(또는 나눠지고 남은 숫자)보다 작거나 같을 동안 반복
    if n % divisor == 0: # 만약 나누어떨어진다면, divisor는 소인수!
      result.append(divisor) # 결과 리스트에 추가!
      n = n / divisor        # 원래 숫자를 이 소인수로 나눠서 다음 계산 준비! (n이 작아짐)
    else: # 나누어떨어지지 않는다면,
      divisor += 1           # 다음 숫자로 나눠보자! (2 다음엔 3, 그 다음엔 4...?)
  return result
```

```
print("\n--- 100을 소인수분해 하면? ---")
print(factorization(100)) # [2, 2, 5, 5] 가 나오겠지?


print("--- 99를 소인수분해 하면? ---")
print(factorization(99)) # [3, 3, 11]
```

궁금이 💡: `while divisor <= n:` 이 부분에서 `n`이 계속 작아지네요! 그리고 `divisor`가 4같은 합성수가 되어도 괜찮나요?

선생님: 아주 예리한 질문이야! 만약 `n`이 4로 나누어떨어진다면, 그 전에 이미 `divisor`가 2였을 때 2로 두 번 나누어졌을 것이기 때문에, `divisor`가 4가 되었을 때는 `n`이 더 이상 4로 나누어떨어지지 않아. 그래서 이 알고리즘은 자연스럽게 소인수들만 찾아내게 된단다!

코딩새싹 🌱: `len()` 함수는 리스트의 길이도 세고, 문자열의 길이도 셀 수 있었죠! 복습!

```
x = [1, 2, 3]
print("\n리스트 x의 길이:", len(x))
my_string = "왕새우만만세"
print("문자열 my_string의 길이:", len(my_string))
```

선생님: (만족스러운 미소를 지으며) 아주 좋아! 오늘 우리는 지난 시간에 배운 소수 판별 함수를 더 깊이 활용해서 소수들 사이의 재미있는 패턴도 찾아보고, 숫자의 근본적인 구성 요소인 소인수들을 찾아내는 강력한 소인수분해 마법까지 배웠어. 이렇게 하나의 개념을 배우면, 그것을 응용해서 점점 더 복잡하고 재미있는 문제를 해결할 수 있게 된단다! 오늘 여러분의 논리력과 탐구심에 큰 박수를 보낸다! 🕵️‍♀️🎉